Behavioral Cloning Project

The goals / steps of this project are the following:

- Use the simulator to collect data of good driving behavior
- Build, a convolution neural network in Keras that predicts steering angles from images
- Train and validate the model with a training and validation set
- Test that the model successfully drives around track one without leaving the road
- Summarize the results with a written report

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Files Submitted & Code Quality

1. Submission includes all required files and can be used to run the simulator in autonomous mode

My project includes the following files:

- model.py containing the script to create and train the model
- drive.py for driving the car in autonomous mode
- model.h5 containing a trained convolution neural network
- writeup_report.md or writeup_report.pdf summarizing the results

2. Submission includes functional code Using the Udacity provided simulator and my drive.py file, the car can be driven autonomously around the track by executing

```
python drive.py model.h5
```

3. Submission code is usable and readable

The model.py file contains the code for training and saving the convolution neural network. The file shows the pipeline I used for training and validating the model, and it contains comments to explain how the code works.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

I used NVIDIA's convolutional neural network model for this project and changed it to get the best performance for my problem. My model consists of five convolutional layers with 5x5 and 3x3 filter sizes and depths between 24 and 64. The model includes RELU layers to introduce nonlinearity, and the data is normalized in the model using a Keras lambda layer. To remove the irrelevant parts of the image such as the sky and the car's trunk, the images are cropped by 65 pixels from the top and 25 pixels from the bottom using a Keras Cropping2D layer.

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting. The original NVIDIA's model contained a dropout layer after every convolution and dens layer. I removed the dropout layers in the fully connected layers and left the dropout layers only after the first, third and fourth convolution layers.

The model was trained and validated on different data sets to ensure that the model was not overfitting. Additionally, the number of epochs was limited to 5 and the training was performed only on a fraction of training data at each epoch (by setting the parameter *steps_per_epoch*) to avoid overfitting. The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually.

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. For the final model, I used a combination of center lane driving, and recovering from the left and right sides for several laps in both directions of the track. My final data set contained 111,411 images before augmentation. For details about how I created the training data, see the next section.

Model Architecture, Training Data Collection and Training Strategy

1. Solution Design Approach

My first attempt was to use the original LeNet convolutional neural network and add some additional convolution and dense layers to the model. To obtain training data, I drove the car in the simulator mode and recorded training data for two laps. In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. I found that my first model had a low mean squared error on the training set but a high mean squared error on the validation set. This implied that the model was overfitting.

To combat the overfitting, I modified the model and added dropout layers after every convolution and dens layer. I noticed that the mean squared error for both training and validation sets was increased significantly. Therefore, I removed the dropouts from the fully connected layers (dens layers) and left them only after the convolution layers in my network. This helped lowering the MSE a little bit. I ran the simulator to see how well the car was driving around track one. There were a few spots where the vehicle fell off the track.

To improve the driving behavior, I collected more data by driving the car in the reverse direction on the track and flipping all images to augment the data. After training my modified LeNet model and testing it in the autonomous mode, I found the model is performing better, but still getting off the track at some point before the bridge.

I went back to the simulator and collected recovery data from the sides of the road for a complete lap. I trained the model again and tested it in the autonomous mode. I noticed the performance has significantly improved and the car can drive smoothly until after the bridge where it leaves the track at the part which has no curb.

I tried playing with the network architecture and tuning its parameters but was not able to improve the performance of the model any further. Therefore, I decided to include side camera images in the training. Due to the large amount of data, I used python generator to generate batches of data during the training process and used Kera *fit_generator* to train the model. For side images, I initially used the correction value of 0.2. After, training the model and testing it in the autonomous mode, I noticed that the car is constantly oscillating to left and right and showing an unsteady behavior. I reduced the correction value to 0.1 and tested the model again. The car performed better but still not as good as when I had not included the side images.

At this point, I decided to try NVIDIA's model. I trained and tested the model with all training data and got a little better performance, but the car was still getting out of the track after the bridge.

Because the performance of my model had not improved after adding the side image data, I decided to remove them. Instead, I collected some more center and recovery data at locations where the car was getting off the road and retrained the model. This attempt significantly improved the performance of my model, and for the first time the car managed to drive autonomously for a complete lap without leaving the track. However, the car was getting close to the sides of the road at a few locations. I played with some training parameters such as the number of epochs and *steps_per_epoch* and managed to reduce the MSE a little bit further. Finally, I tested the model in autonomous mode and found that the car can drive smoothly for a complete lap without getting close to the sides of the road. I obtained the best performance with the following parameter values:

Batch_size = 32
epochs = 5
steps_per_epoch = 200
validation_steps = 20

2. Final Model Architecture

The final architecture of my network is shown the following.

Lambda(lambda x: (x/255) - 0.5, input_shape=(160,320,3))
Cropping2D(cropping=((65,25),(0,0)))

Conv2D(24,(5,5),activation="relu")
MaxPooling2D(2,2)
Dropout(0.25)

Conv2D(36,(5,5),activation="relu")
MaxPooling2D(2,2)

Conv2D(48,(5,5),activation="relu")
MaxPooling2D(2,2)
Dropout(0.25)

Conv2D(64,(3,3),activation="relu")
MaxPooling2D(1,1)
Dropout(0.25)

Conv2D(64,(3,3),activation="relu")
MaxPooling2D(1,1)

Dense(800,activation="relu")
Dense(100,activation="relu")
Dense(10,activation="relu")
Dense(1)